Simplified LQG Control with Neural Networks

Sørensen, O.

Publication date:
1997

Document Version
Også kaldet Forlagets PDF

SIMPLIFIED LQG CONTROL WITH NEURAL NETWORKS

Ole Sørensen

Aalborg University
Inst. of Electronic Systems, Dept. of Control Engineering
Fredrik Bajersvej 7C, DK-9220 Aalborg Ø, Denmark
Phone: +45 96358748, Fax: +45 98151739, E-mail: os@control.auc.dk

Abstract: A new neural network application for non-linear state control is described. One neural network is modelled to form a Kalmann predictor and trained to act as an optimal state observer for a non-linear process. Another neural network is modelled to form a state controller and trained to produce a minimum variance controller for the non-linear process. After training, tuning possibilities for the observer as well as for the controller are introduced to improve the closed loop robustness and noise suppression. The advantage of this method is that tuning takes place after the time consuming training session. The method is illustrated by a simple, multi variable example.

Résumé: Une nouvelle application de réseaux neuraux pour un contrôle non linéaire d'état est décrite. Un predicteur de Kalmann formé à l'aide d'un réseau neural est entraîné pour agir comme un observateur d'état optimal pour un processus non linéaire. Un autre réseau neural modelisant un contrôleur d'état est développé et entraîné comme un contrôleur à variance minimale pour le processus non linéaire. Après l'entraînement, les possibilités d'ajustement et pour l'observateur et pour le contrôleur sont introduites pour améliorer la robustesse et la suppression de bruit du système en boucle fermé. L'avantage de cette méthode est que la procédure d'ajustement n'intervient qu'aprés que le long processus d'entraînement soit expiré. Un exemple à multi-variables est utilisé pour l'illustration de la méthode.

Keywords: Non-linear control, Neural network, Extended Kalmann filters, LQG control, Innovation model.

1. INTRODUCTION

For processes, which are difficult or perhaps impossible to model, the application of neural network control seems very attractive, since a trained neural network is considered to represent a non-parametric model of the process. In most publications a neural network is trained to act as a controller itself, i.e. it is trained to produce the 'best' control signal. Different cost functions produce different 'best' control signals.

(Narendra et al., 1990) minimizes the error between a desired model reference and the output from the process. This method is known from linear control theory as MRAS, Model Reference Adaptive System.

(Psaltis et al., 1988) minimizes the error between the reference and the output from the process. In this way the neural network is trained to act as the inverse process, and a minimum variance controller is obtained.

(Hunt et al., 1992) minimizes a cost function, in which, besides the control error, also the increment of the control signal is 'punished'. This is a generalization to the non-linear case of the LQ-design, known from linear control theory.

The starting point of this paper is the last mentioned category and consequently, the principle of *linear* LQG control, when applying state space description, is illustrated in fig. 1. Accepting, that not all elements of the state $\mathbf{x}_k$ are measurable (k is the discrete sampling number), an observer is introduced giving an estimate $\hat{\mathbf{x}}_k$ of the state, which is fed back as an input to the controller, delivering the control signal $\mathbf{u}_k$ to the process. An output reference $\mathbf{r}_k$ is introduced to handle the servo problem as well.

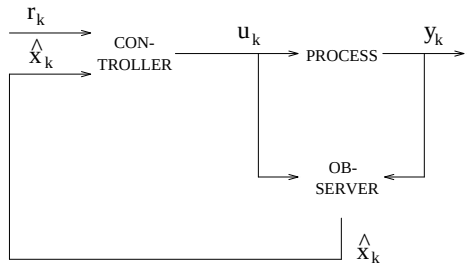

Fig. 1. The principle of optimal control

The observer error is defined as $\mathbf{e}_k = \mathbf{y}_k - \hat{\mathbf{y}}_k$, the control error as $\mathbf{e}_{c,k} = \mathbf{r}_k - \mathbf{y}_k$ and the control signal increment as $\Delta\mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_{k-1}$.

- For the *controller* the performance function
$$V_{con} = E\left[\mathbf{e}_{c,k}^T \mathbf{Q}_1 \mathbf{e}_{c,k} + \Delta\mathbf{u}_k^T \mathbf{Q}_2 \Delta\mathbf{u}_k\right] \quad (1)$$
has to be minimized, using the weight matrices $\mathbf{Q}_1$, 'punishing' $\mathbf{e}_{c,k}$, and $\mathbf{Q}_2$, 'punishing' $\Delta\mathbf{u}_k$, as design parameters. By solving the Riccati equation, a state feedback matrix $\mathbf{L}$ is calculated, giving the optimal controller
$$\mathbf{u}_k = -\mathbf{L}\mathbf{x}_k \quad (2)$$
- For the *observer* the performance function
$$V_{obs} = E\left[\mathbf{e}_k^T \mathbf{e}_k\right] \quad (3)$$
has to be minimized, using the covariance matrix $\mathbf{R}_1$of the process noise and the covariance matrix $\mathbf{R}_2$ of the measurement noise as design parameters. By solving the Kalmann equation, a matrix $\mathbf{K}$ is calculated, giving the optimal observer
$$\begin{aligned} \hat{\mathbf{x}}_k &= \mathbf{\Phi}\hat{\mathbf{x}}_{k-1} + \mathbf{\Gamma}\mathbf{u}_{k-1} + \mathbf{K}\mathbf{e}_{k-1} \\ \mathbf{y}_k &= \mathbf{H}\hat{\mathbf{x}}_k + \mathbf{e}_k \end{aligned} \quad (4)$$
- The *separation theorem* now states, that the optimal control of a process, where not all elements of the state vector are measurable, is the combination of an optimal observer and an optimal controller, giving
$$\mathbf{u}_k = -\mathbf{L}\hat{\mathbf{x}}_k \quad (5)$$

The details of the design method can be found in many textbooks, e.g. (Åström and Wittenmark, 1990) .

If this LQG control method has to be implemented by two neural networks, an observer network and a controller network, to handle non-linear processes as well, two serious drawbacks exist

- Minimizing the performance function (1) is rather complicated, and an off-line method, Back Propagation Through Time (BPTT) has to be applied, (Werbos, 1991) .
- Retuning of the controller demands a quite new training session. Indeed, this is also the case for traditional linear optimal control, but the problem is more perceptible for neural network optimal control, due to the time consuming training session.

Consequently, a heuristic and simplified version of LQG control will be described here by making a radical modification. The performance function to be minimized for the controller is simply changed from (1) to
$$V_{con} = E\left[\mathbf{e}_{c,k}^T \mathbf{e}_{c,k}\right] \quad (6)$$
producing a *minimum variance controller*. After training, a tuning possibility is introduced to re-establish a simplified version of the performance function (1).

The method involves the following four steps.

1. Training the observer, only minimizing the variance of the prediction error, (3).
2. Training the controller, only minimizing the variance of the control error, (6).
3. Tuning the observer by allowing the observed state from the neural network optimal Kalmann predictor to be filtered more or less.
4. Tuning the controller by allowing the control signal from the neural network minimum variance controller to be filtered more or less.

The introduction of the tuning possibilities will be physically interpreted in a way, that emphasizes that this method is a heuristic and simplified version of the traditional LQG control.

Section 2 briefly describes the neural network configurated as a Multi Layer Perceptron (MLP), and defines the so-called Gain matrix. Section 3 describes the neural observer model, the Kalmann predictor, and its training algorithm. Section 4 describes the neural controller model and its training. In section 5 and 6 the tuning possibilities for the observer and the controller, respectively, are explained, allowing tuning after training. In section 7 the method is verified by a practical example controlling a noisy, non-linear, MIMO process.

2. THE MULTI LAYER PERCEPTRON

A Multi Layer Perceptron (MLP) with only one hidden layer has the capability to act as a universal approximator, (Hornik et al., 1989) . An MLP with this structure is in a short matrix notation shown in fig. 2.

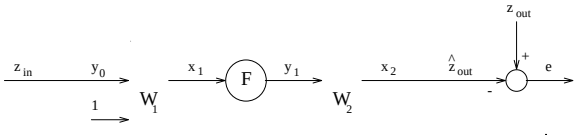

Fig. 2. A one hidden layer MLP in matrix notation

The matrix $\mathbf{W}_1$ represents the weights between the input and hidden layers, where the '1' shown in fig. 2, together with the last column in $\mathbf{W}_1$, produces the necessary offset in the net. The matrix $\mathbf{W}_2$ represents the weights between the hidden and output layers. $\mathbf{F}$ represents a neural vector function with all elements chosen to the tanh-function. The input pattern is represented by the vector $\mathbf{z}_{in}$ and the net output is represented by the vector $\widehat{z}_{out}$. The mismatch between the desired output $\mathbf{z}_{out}$ and $\widehat{\mathbf{z}}_{out}$ is the error $\mathbf{e}$.

Using this notation the output of the net is:

$$\widehat{\mathbf{z}}_{out} = \mathbf{W}_2 \ \mathbf{F}\left(\mathbf{W}_1 \left\{ \begin{array}{c} \mathbf{z}_{in} \\ 1 \end{array} \right\}\right) \tag{7}$$

A training set comprises matched pairs of input $\mathbf{z}_{in}$ and desired output $\mathbf{z}_{out}$.

For static neural networks the most frequently used training method is the Back Propagation Algorithm (BPA) described in many textbooks and papers (Hecht-Nielsen, 1989) , (Rumelhart et al., 1988) , (Narendra et al., 1990) , (Hunt et al., 1992) .

The normal version of BPA uses 'sample-updating', where the individually weights $\boldsymbol{\theta}$ at every sample k are updated in this way

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \eta \frac{\partial V_k(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}_{k-1}} \tag{8}$$

where $V_k(\boldsymbol{\theta})$ is the actual performance function

$$V_k(\boldsymbol{\theta}) = \frac{1}{2}\mathbf{e}_k^T(\boldsymbol{\theta})\mathbf{e}_k(\boldsymbol{\theta}) \tag{9}$$

and η is the gradient factor determining the updating step size.

From a MLP it is possible, during and after training, to extract a so-called Gain matrix $\mathbf{G}$, the derivative of the output vector with respect to the input vector of the net,

$$\mathbf{G} = \frac{d\widehat{\mathbf{z}}_{out}}{d\mathbf{z}_{in}^T} = \mathbf{W}_2 \ \frac{d\mathbf{y}_1}{d\mathbf{x}_1^T} \ \mathbf{W}_1(excl.\ last\ column) \tag{10}$$

Though training of an MLP is considered to produce a non-parametric and non-linear modelling, the above mentioned extraction of the small-signal gain matrix $\mathbf{G}$ allows an estimation of the actual linearized model parameters. Note, that for a non-linear process $\mathbf{G}$ is not a constant matrix, but depends on the actual value of $\mathbf{x}_1$ and consequently on the actual value of the input vector $\mathbf{z}_{in}$.

3. THE OBSERVER

3.1 The Kalmann predictor model

For a *linear* process with m inputs, n states and p outputs the following model of great generality is often used.

$$\begin{aligned} \hat{\mathbf{x}}_k &= \boldsymbol{\Phi}\hat{\mathbf{x}}_{k-1} + \boldsymbol{\Gamma}\mathbf{u}_{k-1} + \mathbf{K}\mathbf{e}_{k-1} \\ \hat{\mathbf{y}}_k &= \mathbf{H}\hat{\mathbf{x}}_k \\ \mathbf{y}_k &= \hat{\mathbf{y}}_k + \mathbf{e}_k \end{aligned} \tag{11}$$

In (11) k is the discrete time, $\hat{\mathbf{x}}$ is the state vector (order n), $\mathbf{u}$ is the input vector (order m), $\mathbf{y}$ is the output vector (order p) and $\mathbf{e}$ is the prediction error vector (order p).

For a linear process $\boldsymbol{\Phi}$, $\boldsymbol{\Gamma}$, $\mathbf{K}$ and $\mathbf{H}$ are constant matrices of dimension $n \times n$, $n \times m$, $n \times p$ and $p \times n$, respectively.

It is assumed, that $\hat{\mathbf{x}}$ is not measurable (Incomplete State Information), and the model is also named the Innovation State Space model.

For the sake of simplicity and without loss of generality the matrix $\mathbf{H}$ here is chosen to $\mathbf{H} = \{\mathbf{I}_{p,p} \ \mathbf{O}_{p,n-p}\}$, where $\mathbf{I}_{p,p}$ is a $p \times p$ unity matrix and $\mathbf{O}_{p,n-p}$ is a $p \times (n-p)$ zero matrix. This means, that the first p elements of the state vector $\hat{\mathbf{x}}$ is filled with $\hat{\mathbf{y}}$.

Given only matched pairs of input measurements $\mathbf{u}_k$ and desired output measurement $\mathbf{y}_k$, $k = 1 \cdots N$, learning this model is equivalent to solving the Extended Kalmann Predictor problem. Even for linear models this is a non-linear problem without any assurance of convergence. In spite of this fact an attempt is made to apply these principles from linear models to be extended to non-linear models.

Generalizing to a *non-linear* Innovation State Space model and including, for convenience, a measurable disturbing vector $\mathbf{d}$, the model reads

$$\begin{aligned} \widehat{\mathbf{x}}_k &= \mathcal{F}\left(\widehat{\mathbf{x}}_{k-1}, \mathbf{u}_{k-1}, \mathbf{d}_{k-1}, \mathbf{e}_{k-1}, \boldsymbol{\theta}\right) \\ \widehat{\mathbf{y}}_k &= \mathbf{H}\widehat{\mathbf{x}}_k, \quad \mathbf{H} = \{\mathbf{I}_{p,p} \ \ \mathbf{O}_{p,n-p}\} \\ \mathbf{y}_k &= \widehat{\mathbf{y}}_k + \mathbf{e}_k \end{aligned} \tag{12}$$

In (12) $\mathcal{F}$ is a non-linear vector function and $\boldsymbol{\theta}$ is a vector containing all weights organized in some structure. In order to train an MLP to learn this model the configuration shown in fig. 3 is applied.

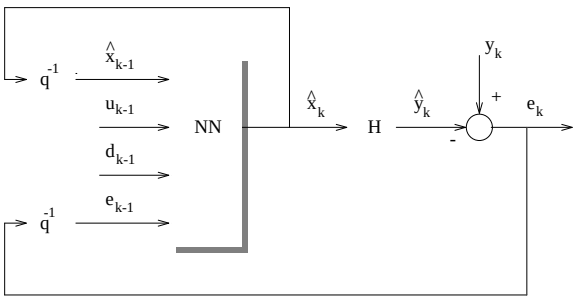

Fig. 3. The observer Kalmann predictor

The output vector is $\widehat{\mathbf{y}}_k = \mathbf{H}\widehat{\mathbf{x}}_k$, the desired output vector is $\mathbf{y}_k$ and the prediction error is $\mathbf{e}_k = \mathbf{y}_k - \mathbf{H}\widehat{\mathbf{x}}_k$. During and after training, the actual Gain matrix $\mathbf{G}_k$ can be extracted from the MLP using (10) and partitioned by

$$\mathbf{G}_k = \frac{d\widehat{\mathbf{x}}_k}{d\mathbf{z}_{in,k-1}^T}$$

$$= \left\{ \frac{\partial \widehat{\mathbf{x}}_k}{\partial \widehat{\mathbf{x}}_{k-1}^T} \; \frac{\partial \widehat{\mathbf{x}}_k}{\partial \mathbf{u}_{k-1}^T} \; \frac{\partial \widehat{\mathbf{x}}_k}{\partial \mathbf{d}_{k-1}^T} \; \frac{\partial \widehat{\mathbf{x}}_k}{\partial \mathbf{e}_{k-1}^T} \right\}$$

$$= \left\{ \hat{\mathbf{\Phi}}_k \; \hat{\mathbf{\Gamma}}_k \; \hat{\mathbf{\Gamma}}_{d,k} \; \hat{\mathbf{K}}_k \right\} \qquad (13)$$

For a non-linear process the actual, linearized parameters $\hat{\mathbf{\Phi}}_k$, $\hat{\mathbf{\Gamma}}_k$, $\hat{\mathbf{\Gamma}}_{d,k}$ and $\hat{\mathbf{K}}_k$ are not constant matrices, but depend on the actual values of the input, state and output vectors.

3.2 Training method

Since fig. 3 comprises two feedback loops around the MLP, it is a Recurrent Network, and training a Recurrent Network is rather more complicated than training a normal 'static' MLP by the Back Propagation Algorithm, see (Hecht-Nielsen, 1989) , (Narendra et al., 1990) , (Billings et al., 1991) .

A second order Recursive Prediction Error Method (RPEM) using a Gauss-Newton search direction has been applied, since this method is dominating in linear system identification.

The method is rather involved and it is explained in several textbooks (Ljung, 1987) and papers (Chen et al., 1990) , (Billings et al., 1991) . In this algorithm the model specific derivative $\boldsymbol{\psi}_k = d\widehat{\mathbf{x}}_k^T / d\boldsymbol{\theta}$, where $\boldsymbol{\theta}$ is a long column vector containing all weights, plays an important role.

Defining

$$\boldsymbol{\psi}_k = \frac{d\widehat{\mathbf{x}}_k^T}{d\boldsymbol{\theta}} \;\; and \;\; \boldsymbol{\phi}_k = \frac{\partial \widehat{\mathbf{x}}_k^T}{\partial \boldsymbol{\theta}} \qquad (14)$$

a calculation of $\boldsymbol{\psi}_k$, using (12), yields

$$\boldsymbol{\psi}_k = \boldsymbol{\phi}_k + \boldsymbol{\psi}_{k-1} \left\{ \hat{\mathbf{\Phi}}_k^T - \mathbf{H}^T \hat{\mathbf{K}}_k^T \right\} \qquad (15)$$

It is strait forward to transform the organizing of the weights in fig. 2 from the two matrices $\mathbf{W}_1$ and $\mathbf{W}_2$ to the long column vector $\boldsymbol{\theta}$ in order to calculate $\boldsymbol{\phi}_k$, and for simplicity this is not shown here.

Below the Recursive Prediction Error Method using a Gauss-Newton search direction is summarized, and for simplicity all the arguments 'θ' are neglected. λ is the forgetting factor.

$$\begin{array}{ll} 1) & Calculate\ \widehat{\mathbf{x}}_k \;\; from \;\; (12) \\ 2) & Calculate\ \boldsymbol{\psi}_k \;\; from \;\; (14) \;\; and \;\; (15) \\ 3) & \mathbf{e}_k = \mathbf{y}_k - \widehat{\mathbf{y}}_k \\ 4) & \mathbf{R}_k = \lambda\, \mathbf{R}_{k-1} + \boldsymbol{\psi}_k\, \boldsymbol{\psi}_k^T \\ 5) & \boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} + \mathbf{R}_k^{-1}\, \boldsymbol{\psi}_k\, \mathbf{E}_k \end{array} \qquad (16)$$

Note, that 1)-2) depend on the chosen specific prediction model, whereas 3)-5) are independent of the model. More practical versions of the algorithm is found in (Ljung, 1987) and (Billings et al., 1991) .

For physical processes the individual elements of the input and output vectors are measured in physical units, which often are of quite different orders of magnitude. A training session on that basis is inconvenient for two reasons. Firstly, noting that the applied neuron functions are tanh-functions, the risk to operate on the near-horizontal parts of the tanh-functions is enlarged, thereby increasing the time consumption for training. Secondly, the training session gives a higher priority to those elements of the output vector, which are accidentally measured in large physical units. For these reasons a scaling is performed, and consequently, training and application of the neural network takes place in a scaled world.

In the rest of this paper all measured signals are given a subscript 'm', contrary to the scaled signals. It is here chosen to scale all measured signals in such a way, that the scaled signals have a mean value of zero and a standard deviation of one.

4. THE CONTROLLER

The functional behavior of the neural controller is, compare to fig. 1.

$$\mathbf{u}_k = \mathcal{G}(\mathbf{r}_{k+1}\,, \widehat{\mathbf{x}}_k\,, \mathbf{d}_k) \qquad (17)$$

For convenience, it is assumed, that the reference signal $\mathbf{r}$ is known one sample ahead, and that a measurable disturbing vector $\mathbf{d}$ is included as an input to the controller.

The purpose of the training is to let the neural network recognize this non-linear relation by minimizing the performance function (6). The training of the controller network (NN2) is performed as a specialized training, (Psaltis et al., 1988) . During this training, the trained observer network (NN1) is used to simulate the non-linear process in order to propagate the control error backward through the simulator to an equivalent error on the output of the controller.

Fig. 4 shows two consecutive samples, unfolded in time, of the closed loop used for training the controller NN2.

For a process with n states and p outputs only the first p states are measurable, since $\mathbf{H} = \{\mathbf{I}_{p,p}\ \mathbf{0}_{p,n-p}\}$. This means, that at each sample k, only desired values for the first p states can be presented, while desired values for the remaining $n-p$ states have to be found from backpropagation of the next coming desired values at sample $k+1$. Consequently, the full state vector $\widehat{\mathbf{x}}_k$ is partitioned into two sub-vectors, the upper p measurable states are denoted $\widehat{\mathbf{x}}_k^1$, and the lower $n-p$ non-measurable states are denoted $\widehat{\mathbf{x}}_k^2$. This partitioning is performed in fig. 4, showing two consecutive sections at sample k and $k+1$.

At each sample $k, k = 1 \ldots N$ the following procedure is performed

1. A forward pass through NN2 and NN1 for section k and $k+1$.

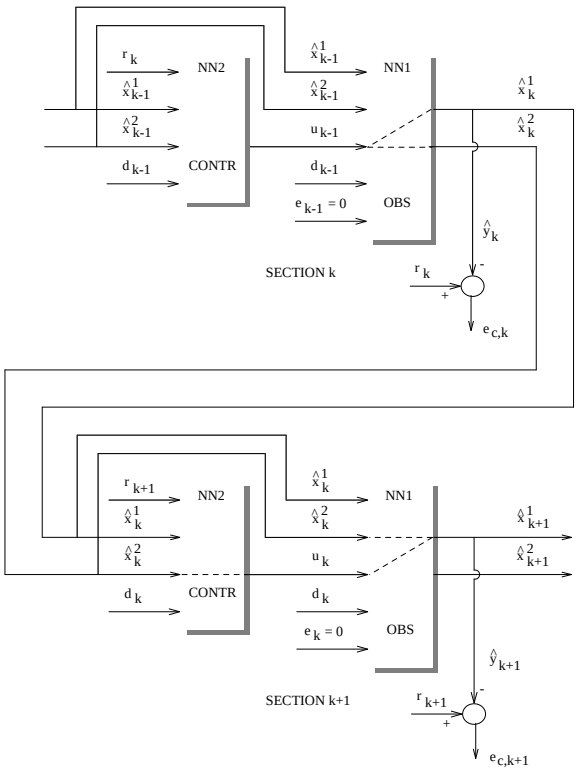

Fig. 4. Two consecutive sections at sample k and $k+1$.

2. The resulting control errors $\mathbf{e}_{c,k} = \mathbf{r}_k - \widehat{\mathbf{y}}_k$ and $\mathbf{e}_{c,k+1} = \mathbf{r}_{k+1} - \widehat{\mathbf{y}}_{k+1}$ are calculated.

3. The corresponding error $\mathbf{e}_{u,k-1}$ on the output of NN2 in section k is calculated by

$$\mathbf{e}_{u,k-1} = \frac{\partial \widehat{\mathbf{x}}_k^T}{\partial \mathbf{u}_{k-1}} \cdot \left\{ \begin{array}{c} \mathbf{e}_{c,k} \\ \mathbf{e}^*_{c,k+1} \end{array} \right\}, where \quad (18)$$

$$\mathbf{e}^*_{c,k+1} = \left\{ \frac{\partial (\widehat{\mathbf{x}}^1_{k+1})^T}{\partial \widehat{\mathbf{x}}^2_k} + \frac{\partial \mathbf{u}_k^T}{\partial \widehat{\mathbf{x}}^2_k} \frac{\partial (\widehat{\mathbf{x}}^1_{k+1})^T}{\partial \mathbf{u}_k} \right\} \mathbf{e}_{c,k+1}$$

This rather complicated expression is easily verified from fig. 4 by checking the backpropagations from $\mathbf{e}_{c,k}$ and $\mathbf{e}_{c,k+1}$ to $\mathbf{u}_{k-1}$. The applied parts of the gain matrices are shown with dashed lines.

4. The weights of NN2 in section k are updated by the Back Propagation Algorithm, using the corresponding error $\mathbf{e}_{u,k-1}$.

The output from the trained controller network NN2, which represents a *minimum variance controller*, is denoted $\mathbf{u}^*_k$.

5. TUNING THE OBSERVER

Tuning the observer is based on the ability of the trained neural Kalmann predictor model also to perform filtering, which now will be explained.

A multiplicative scalar a, $0 \leq a \leq 1$, is placed in the feedback loop from the prediction error, fig. 3, and then the actual linearized model reads

$$\begin{aligned} \widehat{\mathbf{x}}_k &= \hat{\mathbf{\Phi}}_k \widehat{\mathbf{x}}_{k-1} + \hat{\mathbf{\Gamma}}_k \mathbf{u}_{k-1} + \hat{\mathbf{\Gamma}}_{d,k} \mathbf{d}_{k-1} + a \hat{\mathbf{K}}_k \mathbf{e}_{k-1} \\ &= \left\{ \hat{\mathbf{\Phi}}_k - a \hat{\mathbf{K}}_k \mathbf{H} \right\} \widehat{\mathbf{x}}_{k-1} \\ &\quad + \hat{\mathbf{\Gamma}}_k \mathbf{u}_{k-1} + \hat{\mathbf{\Gamma}}_{d,k} \mathbf{d}_{k-1} + a \hat{\mathbf{K}}_k \mathbf{y}_{k-1} \\ \mathbf{y}_k &= \mathbf{H} \widehat{\mathbf{x}}_k + \mathbf{e}_k \end{aligned} \quad (19)$$

The scalar a gives a possibility to change the observer poles, and thereby to change the filter properties of the observer. The extremes are

- a=1. This causes the poles to be placed in the eigenvalues of $\left(\hat{\mathbf{\Phi}}_k - \hat{\mathbf{K}}_k \mathbf{H} \right)$, the optimal Kalmann predictor solution found by training the model. This is equivalent to optimal *prediction*.
- a=0. This causes the poles to be placed in the eigenvalues of $\hat{\mathbf{\Phi}}_k$, the open-loop characteristic matrix. This is equivalent to pure *simulation*.

Choosing a value of a situated between zero and one, allows the observer model to filter more or less.

6. TUNING THE CONTROLLER

The optimal performance function (1) is now simplified to the *actual* performance function V_k, at sample k

$$\begin{aligned} V_k &= (1-p) \frac{1}{2} \mathbf{e}^T_{c,k+1} \mathbf{e}_{c,k+1} + p \frac{1}{2} \Delta \mathbf{u}_k^T \Delta \mathbf{u}_k \\ &= (1-p) V_{1,k} + p V_{2,k} \end{aligned} \quad (20)$$

It is here chosen to 'punish' the control activity by a scalar p, and the control performance by a scalar $1-p$, where $0 \leq p \leq 1$. The extremes are thus $p = 0$ giving *minimum variance* control, and $p = 1$ giving $\Delta \mathbf{u}_k = 0$, i.e. no control action at all.

To find the *actual*, optimal value of the control signal $\mathbf{u}_k$, the gradient of the simplified performance function (20) with respect to $\mathbf{u}_k$ is found

$$\begin{aligned} \frac{dV_k}{d\mathbf{u}_k} &= (1-p) \frac{dV_{1,k}}{d\mathbf{u}_k} + p \frac{dV_{2,k}}{d\mathbf{u}_k} \\ &= (1-p) \frac{\partial \widehat{\mathbf{x}}^T_{k+1}}{\partial \mathbf{u}_k} (-\mathbf{H}^T) \mathbf{e}_{c,k+1} + p \Delta \mathbf{u}_k \\ &= (1-p) \left(\mathbf{u}_k - \mathbf{u}^*_k \right) + p \left(\mathbf{u}_k - \mathbf{u}_{k-1} \right) \end{aligned} \quad (21)$$

This gradient is zero, corresponding to a minimum of V_k, if

$$\mathbf{u}_k = (1-p) \mathbf{u}^*_k + p \mathbf{u}_{k-1} \quad (22)$$

This actual weighting of $\mathbf{u}_k$ between $\mathbf{u}^*_k$ (giving minimum for $V_{1,k}$ alone) and $\mathbf{u}_{k-1}$ (giving minimum for $V_{2,k}$ alone), depending on the choice of p, is illustrated in fig. 5.

Equation (22) is equivalent to calculate $\mathbf{u}_k$ by a low pass filtering of $\mathbf{u}^*_k$

$$\mathbf{u}_k = \frac{1-p}{1-pq^{-1}} \mathbf{u}^*_k \quad (23)$$

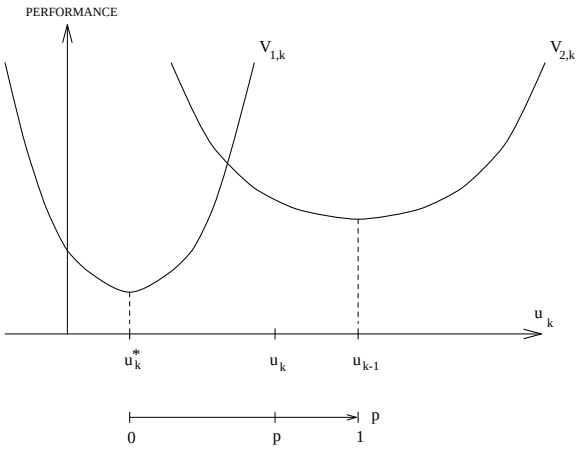

Fig. 5. The actual weighting of $\mathbf{u}_k$ between $\mathbf{u}_k^*$ and $\mathbf{u}_{k-1}$.

The low pass filter has a DC-gain of one and a pole placed in p, $0 \leq p \leq 1$. The extremes are

- p=0. This gives $\mathbf{u}_k = \mathbf{u}_k^*$, i.e. a *minimum variance controller*.
- p=1. This gives $\mathbf{u}_k = \mathbf{u}_{k-1}$, i.e. no control action at all.

The implementation of the total control structure is shown in fig. 6. The observer NN1 is trained *only once* to produce minimum variance of the prediction error, and the controller NN2 is trained *only once* to produce minimum variance of the control error. After training, tuning of the observer is performed by choosing a convenient scalar a, $0 \leq a \leq 1$, and tuning of the controller is performed by choosing a convenient value of the pole p, $0 \leq p \leq 1$ in the low pass filter.

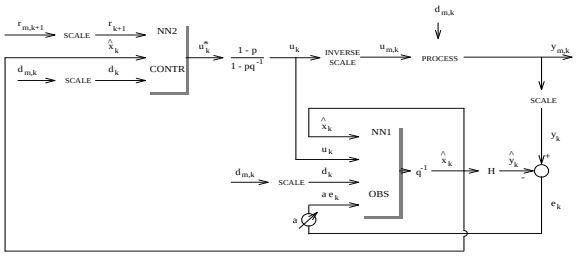

Fig. 6. The implementation of the simplified LQG controller

7. A PRACTICAL EXAMPLE

This closing section deals with a simple practical experiment to illustrate the neural network control concept previously described.

The process considered is a laboratory setup, in which cold and hot water are mixed in a cylindrical tank containing an outlet in the bottom. The control purpose is to maintain the water level and the temperature of the mixed water in spite of several disturbances, mainly the water outlet. The process is equipped with industrial actuators and sensors, but unfortunately they are contaminated with considerably electrical noise. However, the process is accepted as it is. No efforts have been made to establish inner loops around the valves, no filtering of the measured signals are performed, and no knowledge of the process model is assumed. The difficulties are considered to be extra challenges to a neural network controller.

A sketch of the multi variable process is shown in fig. 7, and the process model structure is shown in fig. 8.

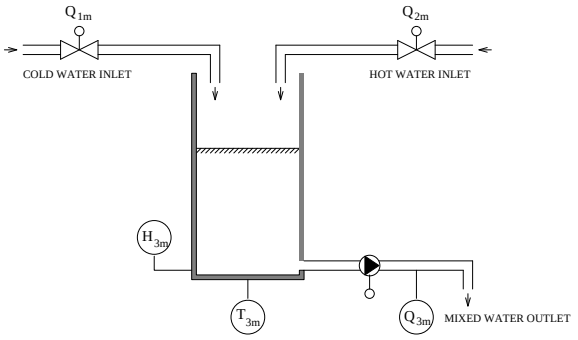

Fig. 7. The multi variable process

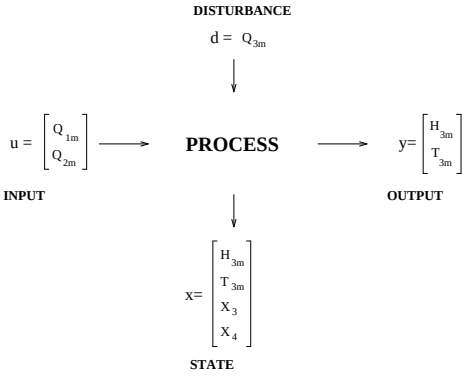

Fig. 8. The process model structure

- The controlling inputs are the voltages Q_{1m} and Q_{2m} to the valves, supplying the cold and hot water, respectively.
- The disturbing input is the outlet flow Q_{3m} of the mixed water. In fact, also the temperatures of the water inlets are disturbances, but they are rather constant and therefore dismissed.
- The measurable outputs are the water level H_{3m} and the temperature T_{3m}. The sensor for T_{3m} is rather slow, it contains a time constant of approximately ten seconds.
- The state is chosen to fourth order, the first two states being the measurable outputs H_{3m} and T_{3m}, and the last two states leaving 'space' for unknown actuator

and sensor dynamics.

- The references are chosen to $0.25\ m$ for H_{3m} and $25\ C$ for T_{3m}.
- The sampling interval is $2\ sec.$

Without any knowledge of the process model, two traditional PI-controllers are tuned 'by hand' to give the 'best' performance, letting one PI-controller control H_{3m} by Q_{1m}, and the other PI-controller control T_{3m} by Q_{2m}. The result is shown in fig. 9.

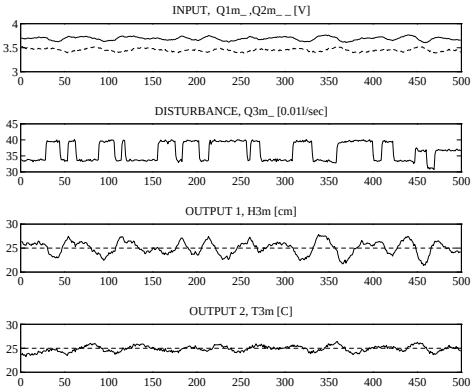

Fig. 9. Two individually tuned PI-controllers. The data are used for training the neural networks.

The unknown hidden coupling between the two individual control loops causes a rather poor performance. Both control loops try to fulfill its own purpose, regardless of the other loop.

The data from fig. 9 are used as training data for the neural networks. The number of hidden neurons is 8 for the observer and 6 for the controller. Fig. 10 shows the closed loop result after training and tuning with $p = 0.1$ and $a = 0.3$.

Considering the considerable disturbance and noise, a nice performance is obtained. An eventual retuning of the controller may be performed easily without any retraining.

8. CONCLUSION

A new neural network application for non-linear state control was described. One neural network was modelled to form a Kalmann predictor and trained to act as an optimal state observer for a non-linear process. Another neural network was modelled to form a state controller and trained to produce a minimum variance controller for the non-linear process. After training, tuning possibilities for the observer as well as for the controller were introduced to improve the closed loop robustness and noise suppression. The advantage of this method was that tuning took place after the time consuming training session. The method was illustrated

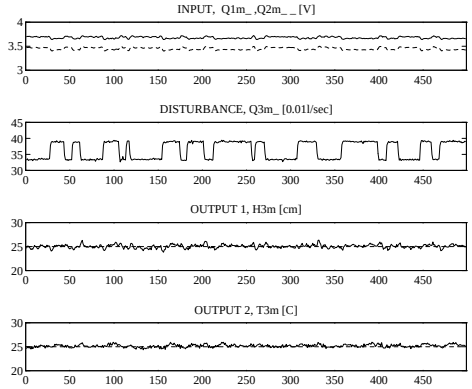

Fig. 10. Simplified optimal control with $p = 0.1$ and $a = 0.3$.

by a simple multi variable example.

The result from the experiment confirmed the practical application of the control and training concepts.

REFERENCES

Billings, S. A. et al. (1991). A comparison of the backpropagation and recursive prediction error algorithms for training neural networks. *Mechanical Systems and Signal Processing*, 3:233–255.

Chen, S. et al. (1990). Non-linear system identification using neural networks. *Int. J. Control*, 6:1191–1214.

Hecht-Nielsen, R. (1989). *Neurocomputing*. Addison Wesley.

Hornik, K. et al. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366.

Hunt, K. J. et al. (1992). Neural networks for control systems - a survey. *Automatica*, 6:1083–1120.

Ljung, L. (1987). *System identification, theory for the user*. Prentice-Hall, first edition.

Narendra, K. S. et al. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transaction on Neural Networks*, 1(1):4–27.

Psaltis, D. et al. (1988). A multilayererd neural network controller. *IEEE Control System Magazine*, 8(3):17–21.

Rumelhart, D. E. et al. (1988). *Parallel Distributed Processing*, volume 1,2. MIT Press, Cambridge, London.

Åström, K. J. and Wittenmark, B. (1990). *Computer Controlled Systems: Theory and Design*. Prentice-Hall, second edition.

Werbos, P. J. (1991). Backpropagation through time: What it does and how to do it. In *Neural Networks, Theoretical Foundations and Analysis*, pages 211–221. Clifford Lau, IEEE Press.